# 1. Polynomial

## 1.1. Aim

The aim of this checkpoint to write a Python class to represent a polynomial and to use this class to perform simple mathematical operations on polynomials.

Note

The code for this checkpoint **must not** make use of external libraries, such as the `numpy.polynomial` module, that can automatically perform many of the required operations.

## 1.2. Checkpoint task

Write a Python class to represent a polynomial of the form:

$$P(x) = \sum_{i=0}^{n} a_i x^i = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

where the coefficients $a_i$ are real numbers.

In the task description below we have assumed that the class name is `Polynomial`.

Your class should include:

- A `list` instance variable to hold the coefficients of $P(x)$.
- An `__init__()` method to set the values of the coefficients $a_i$, which should be supplied as a `list`.
- Methods to:
  1. Calculate and return the order of $P(x)$
  2. Add another Polynomial object to $P(x)$ and return the result as a new `Polynomial` object (it is not sufficient to return the result as a `list` of coefficients). Your code should include the case where the polynomials being added are of different order.

     ▶ Example: adding polynomials

  3. Calculate the derivative of $P(x)$ and return the result as a new `Polynomial` object.

     ▶ Differentiating polynomials

  4. Calculate the antiderivative (indefinite integral) of $P(x)$ and return the result as a new `Polynomial object`. A numerical value for the constant of integration should be supplied to the method.

     ▶ Integrating polynomials

  5. Print a sensible *string* representation of `P(x)`, for example in the form:

     ```
     P(x) = a_0 + a_1 x^1 + a_2 x^2 + .... + a_n x^n
     ```

     For full credit, the string should:

     - not print "x^0" with the first term:

       ```
       P(x) = 1.0 + 3.0 x^2
       ```

       NOT:

```
P(x) = 1.0 x^0 + 3.0 x^2
```

- skip terms where $a_i = 0$:

```
P(x) = 1.0 + 3.0 x^2
```

NOT:

```
P(x) = 1.0 + 0.0 x^1 + 3.0 x^2
```

All the methods should be *instance methods* (rather than class methods or static methods).

You should also write a test code to check that your class works. This should create two `Polynomial` objects to represent the following polynomials:

- $P_a(x) = 2 + 4x^2 - x^3 + 6x^5$
- $P_b(x) = -1 - 3x + 4.5x^3$

and then:

1. Calculates the order of $P_a(x)$
2. Adds $P_b(x)$ to $P_a(x)$
3. Calculates the first derivative of $P_a(x)$
4. Calculates the antiderivative of this result, i.e. the antiderivative of $dP_a(x)/dx$. The constant of integration $c$ should be set to $c = 2$.

▶ Check you result!

In each case your code should print the results to the screen, using your *String* representation of a `Polynomial` object where appropriate.

**Keypoint 1.2.1**

Note: As this checkpoint is concerned primarily with writing and using classes, your test code does not need to take input from the terminal.

## 1.3. Marking Scheme

- Marking Scheme for Polynomial Checkpoint

Note!

The code you submit for assessment **must be your own work**, not the output of Generative AI or the work of another person. The course team will use their best judgement to determine whether this is true. See the discussion in the Checkpoint Marking Scheme and the Generative AI Policy on the course Learn page for more information.

## 1.4. Optional extras

*Optional extras are not marked and don't count towards the assessment for the checkpoint.*

If you have the time, here is another exercise that employs basic OO functionality: write a class to generate (pseudo-)random numbers, given a suitable 'seed'.

A simple algorithm you can use to generate random numbers is:

```
seed = float(seed * a + c) % m
random_number = abs(seed / m)
```

where $m = 233280, a = 9301, c = 49297$ and `seed` is an integer.

Write a test class that prints 10 pseudo-random integers (between 0 and a specified maximum value) to the screen, using the methods in your random number generator class.

A common way of seeding the RNG is to use an integer such as your birthdate. This will produce the *same* list of 10 random numbers each time it is run. Why produce random numbers that are not random? Well, sometimes this is useful as a debugging tool or a way of testing the influence of changing only one parameter in a “randomly” initialised simulation.

Alternatively, more “randomness” can be inserted using the current time cast as an integer. Importing the module `time` and calling the `time.time()` method (which returns the current time as floating point seconds since the ‘epoch’ – 12:00am, January 1, 1970) will enable you to do this. This method *should not* produce the same list of random numbers on subsequent runs.

At the end of all this, you may be dismayed (or relieved) to note that Python has built-in functionality for random number generation that is a lot more random than this simple algorithm : the method *random()* in the module *random*.

▶ More random numbers

## 1.5. Relevant course sections

Studying the following course sections will help you complete this checkpoint:

- Introduction to object-oriented programming
- Classes, Objects and Methods

Additional material that you may find useful:

- Magic methods